

Lab: Create Feedback Form for Survey



Estimated time needed: 40 minutes

What you will learn

In this lab, you will create a feedback form using React functional components and manage user details using the `useState` hook. You will implement event handlers to manage form input changes, validate user inputs, and handle form submissions. Additionally, you will create a confirmation dialog using the `confirm` method to confirm user details before final submission. Upon successful submission, you will reset the form fields and display a thank you message to the user. This lab will give you practical experience building interactive forms and handling user inputs in React applications.

Learning objectives

After completing this lab, you will be able to:

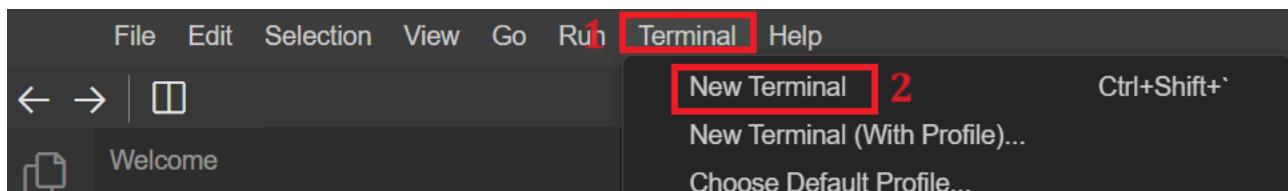
- Create a form to collect user feedback, including their name, email, and the feedback message.
- Handle form submission, including data validation, to ensure that the users enter their names and provide feedback before submitting the form.
- Display a confirmation dialog to users after they submit their feedback, showing the information they entered and prompting them to confirm before final submission.
- Reset the form fields to clear the input values and provide users with a clean form for submitting additional feedback.

Prerequisites

- Basic knowledge of HTML
- Intermediate knowledge of JavaScript
- Basic knowledge of React function component, `useState` hook, and handling forms

Step 1: Setting up the environment

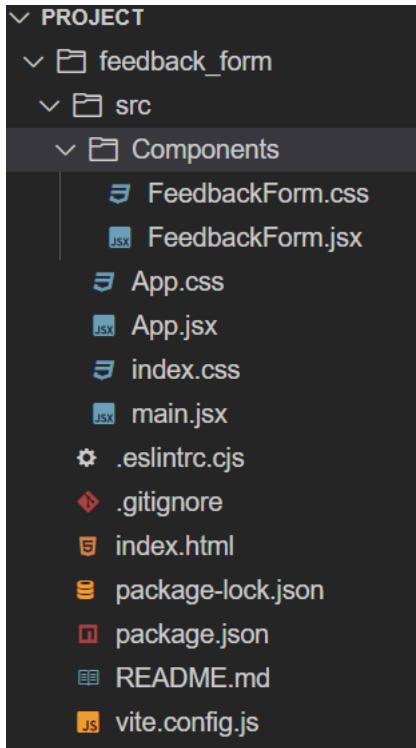
1. From the menu on top of the lab, click the **Terminal** tab at the top-right of the window shown at number 1 in the given screenshot, and then click **New Terminal** as shown at number 2.



2. Write the following command in the terminal and hit Enter to clone the boiler template for this React application.

```
git clone https://github.com/ibm-developer-skills-network/feedback_form.git
```

3. This will create a folder named **feedback_form** under the project folder name, and the structure will be as shown in the given screenshot. The **feedback_form** application includes a class component named **FeedbackForm.jsx** and a css file named **FeedbackForm.css**.



4. Write the command to enter the **feedback_form** folder in the terminal. The command will set your terminal path to run the React application in the **feedback_form** folder.

```
cd feedback_form
```

5. To make sure that the code you have cloned is working correctly, you need to follow the given steps:

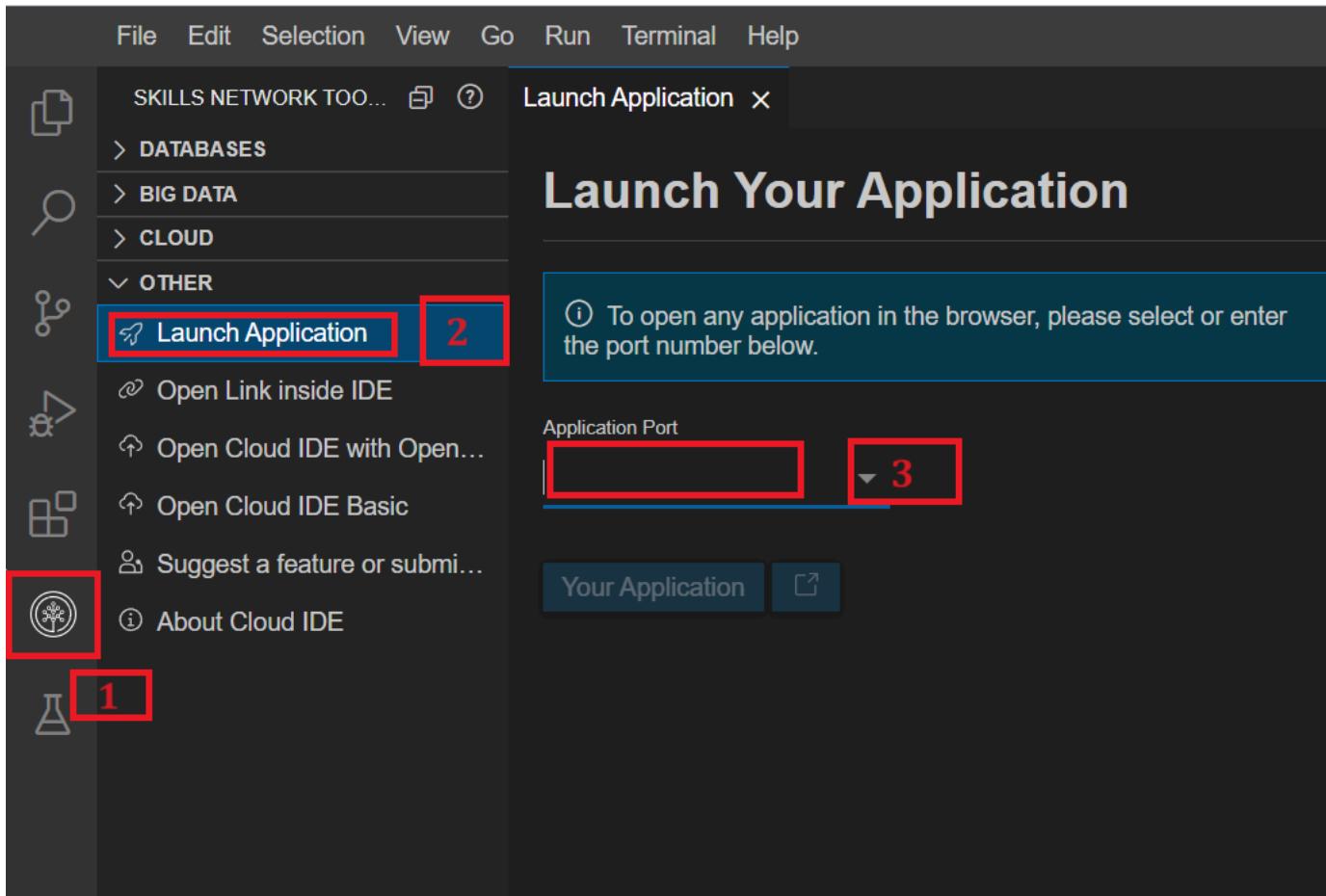
- Write the following command in the terminal and select Enter to install all the necessary packages to execute the application.

```
npm install
```

- Then execute the following command to run the application, providing you with port number 4173.

```
npm run preview
```

6. To view your React application, click the Skills Network icon on the left panel (refer to number 1). This action will open the **Skills Network Toolbox**. Next, click **Launch Application** (refer to number 2). Enter port number **4173** in **Application Port** (refer to number 3) and click .



7. The output will display as shown in the given screenshot.

Tell Us What You Think

We'd Love to Hear From You!

Please share your feedback with us.

8. You can preserve your latest work on this lab by adding, committing, and pushing it to your GitHub repository. This ensures that even if you're not working on the task continuously, your progress will be saved, allowing you to resume from where you left off.
Note: Step 8 is optional.

Step 3: Create basic template

1. Navigate to the **Components** folder in your React project's src folder.
2. Navigate to the **FeedbackForm.jsx** component located within the src folder of your React project. Inside the `return` of this component, you have a `<nav>` tag and a `<form>` tag with class name `feedback-form`, which includes a child `<h2>` tag with a `<p>` tag.
3. To create the feedback form, you need to include three input fields and create one button to handle the user's following details.
 - o First input box for username
 - o Second input box for user email ID
 - o Third input box for user feedback

```

<input
  type="text"
  name="name"
  placeholder="Your Name"
/>
<input
  type="email"
  name="email"
  placeholder="Your Email"
/>
<textarea
  name="feedback"
  placeholder="Your Feedback"
></textarea>
<button type="submit">Submit Feedback</button>

```

4. To see the latest output, perform **ctrl+c** and re-run the application in the terminal. You will see the output in the screenshot.

The screenshot shows a feedback form with the following structure:

- Title:** We'd Love to Hear From You!
- Text:** Please share your feedback with us.
- Input Fields:**
 - Text input field labeled "Your Name"
 - Text input field labeled "Your Email"
 - Text area labeled "Your Feedback" with a scroll bar.
- Button:** A red button labeled "Submit Feedback".

Note: If application is already running then you just need to refresh the application page

Step 4: Initialize states to handle form data

1. Integrate the useState hook from React at the top of the component to manage the form data state. Use the useState hook to create state variables for the form data details such as name, email, and feedback.

```
import React, { useState } from 'react';
```

2. Initialize the details with empty strings. You need to initialize an object, including name, email, and feedback form details, using the **useState** hook.

```
const [formData, setFormData] = useState({
  name: '',
  email: '',
  feedback: ''
});
```

Note: Include the above code before return of this component.

Step 5: Implement change handlers

1. Define a **handleChange** function to update the form data state as users input their information. Create this after you have initialized variables with **useState** hook.
2. Inside the **handleChange** function, extract the name and value of the input field from the event object.
3. Update the form data state using the **setFormData** function and spread operator to merge the new value with the existing form data.

```
const handleChange = (event) => {
  const { name, value } = event.target;
  setFormData({
    ...formData,
    [name]: value
  });
};
```

- `const handleChange = (event) => { ... };`: This line declares a constant named `handleChange` and assigns it an arrow function. The function takes an event parameter, representing the event triggered by the user's interaction with an input element.
- `const { name, value } = event.target;`: This line extracts the name and value properties from the event object's target property. The target property refers to the DOM element that triggered the event, which, in this case, is an input field. The name property corresponds to the 'name' attribute of the input field, while the value property represents the current 'value' entered by the user into the input field.
- `setFormData({ ...formData, [name]: value })`: The **setFormData** function is a state updater function provided by React's **useState** hook to update the state variable `formData`. It spreads the existing `formData` object and then updates the property specified by the 'name' variable with the new value. This pattern ensures that the state is updated immutably, meaning a new object is created with the updated property rather than modifying the existing state directly.

Step 6: Integrate form state and onchange event

1. In the JSX code, bind the form input fields, name, email, and feedback to their respective state variables using `formData.name`, `formData.email`, and `formData.feedback`.
2. Use the **value** attribute to set the input field values and the **onChange** attribute to call the **handleChange** function when the input values change.

```
<input
  type="text"
  name="name"
  placeholder="Your Name"
  value={formData.name}
  onChange={handleChange}
/>
<input
  type="email"
  name="email"
  placeholder="Your Email"
  value={formData.email}
  onChange={handleChange}
/>
<textarea
  name="feedback"
  placeholder="Your Feedback"
  value={formData.feedback}
  onChange={handleChange}
></textarea>
```

Note: Include value and onchange attributes in your previous input field and text area.

Step 7: Handle form submission

1. Implement the form submission functionality by defining a **handleSubmit** function. This function should take an event parameter and prevent the default form submission.
2. Then, create a variable named **confirmationMessage** to capture user details.
3. Next, create another variable, **isConfirmed**, to confirm whether the user's details are correct.

4. If confirmed, log the form data to the console, display a **thank you** message to the user using the alert box, and clear the form fields by resetting the form data state.

```
const handleSubmit = (event) => {
  event.preventDefault();
  const confirmationMessage = `
    Name: ${formData.name}
    Email: ${formData.email}
    Feedback: ${formData.feedback}
  `;
  const isConfirmed = window.confirm(`Please confirm your details:\n${confirmationMessage}`);
  if (isConfirmed) {
    console.log('Submitting feedback:', formData);
    setFormData({
      name: '',
      email: '',
      feedback: ''
    });
    alert('Thank you for your valuable feedback!');
  }
};
```

- This code contains:

- o `const confirmationMessage = ...;`: This template constructs a confirmation message using the data from the `formData` object. It includes the name, email, and feedback fields the user enters.
- o `const isConfirmed = window.confirm(...);`: This line displays a confirmation dialog using the `window.confirm()`, presenting the `confirmationMessage` to the user. If the user confirms the details in the dialog, `isConfirmed` is set to true; otherwise, it's set to false.
- o `if (isConfirmed) { ... }`: This conditional statement checks if the user has confirmed their details by clicking "OK" in the confirmation dialog.
- o `setFormData({ ... })`: The `setFormData` function is called to reset the `formData` state to empty values, clearing the form fields after submission.
- o `alert('Thank you for your valuable feedback!');`: After resetting the form, an alert is displayed to the user thanking them for their feedback.

Note: Include this code just before the return of the `FeedbackForm` component.

Step 8: Implement onSubmit event handler

- To submit the details, include the `onSubmit` event handler in the `<form>` tag.

```
<form onSubmit={handleSubmit} className="feedback-form">
```

Step 9: Check the output

- Now, you need to rerun the application again and fill the details. Then click **Submit Feedback**. A confirm box will appear with the filled details. If application is already running in browser, then just refresh the page.

Please confirm your details:

Name: John
 Email: john@example.com
 Feedback: The product was great



- If you click **OK**, it will display another alert box with a thank you note.

Thank you for your valuable feedback!

OK

3. If you click **Cancel** in the confirm box, the form with filled details will reappear.

Congratulations! You have created a feedback form application!

4. ► Click here for the entire code solution for FeedbackForm.jsx

Practice Exercise

1. In this exercise you will create a rating system using radio buttons to let users provide rating as well.

2. First you need to create one variable named `rating` within the `formData useState` hook and initialize it as empty string.

Hint: Include this variable as key value pair within `formData`

- Click here for the solution

3. For this you need to use input box with type radio button. To create a rating system using **five** radio buttons, where each radio button represents a rating value from 1 to 5. Each radio button should display its corresponding value: the first radio button showing “1”, the second showing “2”, and so on up to the fifth radio button displaying “5”. Make sure that input box with type radio button have `onChange` event handler to update the `formData` state to reflect the new selection made by the user using `handleChange` function.

Hint: Use input box filed with `type='radio'`. Call event handler function using `onChange={handleChange}`

- Click here for the solution

4. `handleChange` function will remain unchanges as you are taking the entire information using `...formData` spread operator.

5. Now you need to incorporate the rating details in the `handleSubmit` function. For this you need to take user selected value for rating from `formData` vraibale and include it in `confirmationMessage` to display along with other details.

Hint: Put `formData.rating` value in one variable to access and display user input for radio button filed

- Click here for the solution

6. Now check the output by re-running the application again. The output will look like according to given screenshot.

Tell Us What You Think

We'd Love to Hear From You!

Please share your feedback with us.

Rate Us:

 1 2 3 4 5Submit Feedback

7. Fill the entire form again and check the alert message. It will display rating along with other details as well.

Please confirm your details:

Name: John

Email: john@example.com

Feedback: It was great

Rating: 5

OKCancel

► Click here for the entire code solution

Conclusion

- You have implemented a form to collect user feedback in the given 'FeedbackForm' component. The component utilizes React's useState hook to manage form data state, including the user's name, email, and feedback.
- The 'handleChange' function updates the form data state as the user inputs information into the form fields. Upon form submission, the 'handleSubmit' function prevents the default form submission behavior, displays a confirmation dialog with the user's details, clears the form fields, and shows a thank you message upon confirmation.
- The form includes input fields for the user's name and email, and a text area for providing feedback.

Author(s)

Richa Arora

